AMENDMENTS TO THE SPECIFICATION

Amendments to the Specification are identified by the paragraph numbers of the present application as published on June 24, 2004 under United States Patent Application Publication No. 2004/0123223.

Please replace paragraph 0001 with the following:

[0001] The present application claims the benefit of United States Provisional Patent Application Nos. 60/399,635 entitled "Data Dispersion and Mirroring Method with Fast Dual Erasure Correction and Multi-Dimensional Capabilities" filed on July 29, 2002 and 60/460,545 entitled "Composite Data Protection Method for Micro Level Data" filed April 4, 2003.

Please replace paragraphs 0019 and 0020 with the following:

[0019] The prior art includes some solutions based on the dispersal of data with the recovery via dual erasure control coding. In the context of small disk arrays this is commonly referred to as method RAID level 6. The number of disk drives clustered in these arrays typically varies from as few as four to six to as many as nine or seventeen. Other erasure control applications may be called information dispersal and reconstruction, erasure or loss resilient coding or in specific context forward error correction coding, FEC.

[0020] U.S. Pat. Nos. 5,128,810 (Halford) and 5,283,791 (Halford), in the name of the present inventor, show dual drive error correction method using a data bit dispersal with erasure control technique. This method requires a parity side-band drive and a significant amount of computation due to the dual dependent ECC codes (odd parity and irreducible polynomial). This approach is relatively cumbersome and slow if implemented in software. Also the code data array is rectangular M * (M-1) which limits data size flexibility.

Please replace paragraphs 0115 and 0116 with the following:

[0115] Consider a dual-element data set [st] conventionally mirrored by two additional data sets [uv] and [qr]. A system with this much redundancy must anticipate more than two potential failures before corrective reconstruction can be scheduled. Consider the effects of three and four failures. Clearly five failures within a six element system results in a catastrophic loss of data. For conventional triplicate redundancy there are two combinations of three failures out of the 20 possible combinations that would result in a catastrophic loss of data. They are [suq] and [tvr]. Likewise there are 6 combinations of four failures out of the 15 possible combinations that would also result in a catastrophic loss of data. They are [tvqr], [tuvr], [suqr], [suvq], [stvr] and [stuq]. Simple triple redundancy appears to have substantial mathematical deficiencies with eight failing combinations.

[0116] The invention can disperse rows q, r, s, t, u and v and recover all but three combinations of 1, 2, 3 and 4 element failures. Random dual bit error correction and data validation is possible as with the inventions four element dispersal method. It is also an object of the invention to permit dispersal of 4-bit column elements m, n, o and p.

Please replace paragraph 0122 with the following:

[0122] However if two of the elements among "s", "t" and "u" are good then there must also be two good element pairings among "sv", "tv" and "uv" that decode data correctly, agree on the same data byte and indicate a common and correct "v" element. Since we don't know "v" we must evaluate all 16 possible data decodes for "sv", "tv" and "uv" with "v" ranging from 0 to F hexadecimal. For line speed applications this must be evaluated simultaneously.

Please replace paragraph 0128 with the following:

[0128] A further object of the invention is that the codeword packet can be developed over time and stored in dispersed locations and on disparate media. For instance the data bytes could be stored on disk drives for fast random access performance and the ECC bytes could be on tape. Another strategy is to have the data bytes on disk and the complete codeword distributed across 4 tape cartridges. Reading any two tape cartridges successfully would recover data bytes lost on tape or disk. Backup to tape could occur overnight when the disk files see potentially less activity and snapshots of data are easier.

Please replace paragraphs 0130 and 0131 with the following:

[0130] While the present invention consistently shows a data byte distributed equally across all four elements of an array it is not the intention of the invention to restrict higher level hardware, software or firmware from distributing records, sectors, files or data bases onto individual elements of the array. The invention simply makes it possible to distribute each byte over the array for security concerns.

[0131] Another object of the invention is that data can be dispersed to four devices then during read back the four data segments are assessed as to which devices need recovery and whether the error correction or the erasure recovery method should be used. Also it may be advantageous that device groups can fluctuate between two and four units as resiliency requirements change. In a virtualized system all devices can be non-arrayed and the application or file system decide on the resiliency level and choose to disperse data to two, three or four devices. Also it may be advantageous to migrate data first to two devices, one containing data and the other ECC bytes then later migrate data to "s" and "t" devices and ECC to "u" and "v" devices.

Please replace paragraph 0164 with the following:

[0164] Once the data packet exits the CRC Checker, item 705, it is buffered in the Data Buffer In logic block, item 700. If no data packet CRC error is detected at the node input the buffered data packet can be moved into the node, item 760. Here again it is prudent to reverify the CRC as the data packet can possibly be corrupted in the buffering process. The data passes from data buffer, item 700 to the node, item 760, through the second CRC Checker, item 730. If a CRC error is detected the Message Control In logic block, item 715, is again signaled to reply to the transmitting node with a NACK signal. A CRC error occurring at either CRC Checker will flag the data packet to be in error at the node, item 760. If no CRC errors have been detected the Message Control In logic block, item 715 replies to the transmitting node with an Acknowledge signal.

Please replace paragraph 0166 with the following:

[0166] The explanation block 795 details the encoding of ACK and NACK signals. Effectively there is no special encoding at this level, an active ACK line translates as an ACK and an active NACK line translates as a NACK. Setting both ACK and NACK is not applicable and disallowed.

Please replace paragraph 0205 with the following:

[0205] Step 1285 sends ACK and NACK signals requesting a retry of the previous data packet then advances to step 1225.

Please replace paragraph 0212 with the following:

[0212] Referring now to Figure 19, thereshown is flowchart of process for encoding codeword arrays. After starting at block 300, a Message Byte Count is set to zero in block 305. Next, at block 310, a Next Data Byte [D] is retrieved. Then, at block 315, Table 1 is addressed with Data Byte [D] and a codeword is loaded. An optional step of generating separate Cyclic Redundancy Check (CRC) for each Data Byte may be performed at block 320. Then, at block 325, data buffers are aggregated for dispersion of the packet across multiple channels (here 4). The Message Byte Count is then incremented at block 330. The process then determines at block 335 whether an entire message has been encoded. If no, the process cycles back to block 310. If yes, the process determines whether a CRC is to be performed at block 340. This process is optional. If no CRC is to be performed, the process stops at block 345. If a CRC is to be performed, the process continues on at block 350 by encoding CRC bytes in the same manner as the data bytes in block 350. Table 1 is then addressed with the CRC byte and loaded with a codeword array at block 355. Again, data buffers are aggregated for dispersion of the codeword array elements across multiple channels in block 360. The CRC count is then incremented in block 365 and the process determines if it has reached the end of the CRC process. If not, the process loops back to block 350. If so, the process ends at block 375.